Express JS

1. **What is express JS? What are the concept of Express?**

   →Express.js is a popular web application framework for Node.js that simplifies the process of building robust and scalable web applications. It provides a wide range of features and concepts to facilitate routing, middleware handling, and server-side development. Here are the key concepts in Express.js:

   - **Routing:** Express.js allows you to define routes for handling HTTP requests. Routes can be created using HTTP methods (GET, POST, PUT, DELETE, etc.) and path patterns. Route handlers are functions that are executed when a specific route is matched.

   - **Middleware:** Middleware functions in Express.js are functions that have access to the request and response objects and can modify them or execute additional logic. Middleware can be used for tasks such as authentication, request parsing, logging, error handling, etc. Middleware functions can be mounted globally or specific to certain routes.

   - **Request and Response Objects:** Express.js provides request (req) and response (res) objects to handle incoming requests and send responses back to the client. These objects contain properties and methods to access request headers, parameters, query strings, request bodies, and to send response data and set response headers.

   - **Route Parameters:** Express.js allows you to define route parameters in the URL path. Route parameters are denoted by a colon followed by the parameter name (/users/:id). These parameters can be accessed within route handlers using req.params.

   - **Query Parameters:** Query parameters are used to send data in the URL as key-value pairs (/users?id=123). Express.js provides access to query parameters through req.query object.

- **Static Files:** Express.js can serve static files such as HTML, CSS, JavaScript, images, etc. You can define a static file directory using express.static() middleware, allowing the server to directly serve these files without any additional routing.

- **Error Handling:** Express.js provides mechanisms to handle errors in middleware and route handlers. Middleware functions can use next() to pass errors to the next error-handling middleware. Additionally, you can define a special error-handling middleware that catches errors and sends appropriate responses.

- **View Engines:** Express.js allows you to use view engines to generate dynamic HTML templates. Popular view engines like EJS, Pug (formerly Jade), and Handlebars can be integrated with Express.js to render dynamic views and templates.

- **Routing Middleware:** Express.js provides a way to modularize your routes using the express.Router object. With this feature, you can create separate route files and mount them on specific paths within your application.

- **Server Configuration:** Express.js provides methods to configure server settings such as port, hostname, and various server options. It allows you to start the server with app.listen() and handle server events like listening and error.

```
const express = require('express');
const app = express();
const port = 3000;

// Middleware for parsing JSON
app.use(express.json());

// Static files middleware
app.use(express.static('public'));

// Custom middleware for logging
app.use((req, res, next) => {
  console.log(`[${new Date().toISOString()}]
${req.method} ${req.url}`);
```

```
  next();
});

// Route with route parameters
app.get('/users/:id', (req, res) => {
  const userId = req.params.id;
  // Logic to fetch user by ID from the database
  res.send(`Fetching user with ID: ${userId}`);
});

// Route with query parameters
app.get('/search', (req, res) => {
  const searchTerm = req.query.q;
  // Logic to perform search based on the query term
  res.send(`Search results for: ${searchTerm}`);
});

// Error handling middleware
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).send('Something went wrong!');
});

// Route for rendering dynamic views
app.set('view engine', 'ejs');
app.get('/products', (req, res) => {
  const products = ['Product 1', 'Product 2', 'Product
3'];
  res.render('products', { products });
});

// Router middleware
const userRouter = express.Router();
userRouter.get('/', (req, res) => {
  res.send('User home page');
});
userRouter.get('/profile', (req, res) => {
  res.send('User profile page');
});
app.use('/users', userRouter);

// Server configuration
app.listen(port, () => {
```

```
  console.log(`Server is running on port ${port}`);
});
```

- We set up middleware for parsing JSON using express.json() to handle JSON data in request bodies.
- We use the static files middleware with express.static() to serve static files from the "public" directory.
- Custom middleware is implemented to log the incoming requests, showing the method and URL.
- The route /users/:id demonstrates the usage of route parameters. It retrieves the user ID from the route parameters using req.params.
- The route /search showcases the usage of query parameters. It retrieves the search term from the query string using req.query.
- Error handling middleware is defined using app.use() to handle and log errors that occur in the application.
- Dynamic views are rendered using the EJS view engine. The /products route renders the "products.ejs" template and passes an array of products to be rendered.
- Router middleware is used to modularize routes. The userRouter handles routes related to users and is mounted on the /users path using app.use().
- The server is configured to listen on port 3000 using app.listen().

This example demonstrates the use of routing, middleware, request and response objects, route parameters, query parameters, static files serving, error handling, view engines, routing middleware, and server configuration in an Express.js application.